DGMD E-17: Homework Assignment A4

1. Assignment Details

Assignment 4
Object Tracking Drone Application Using OpenCV, MAVSDK and PX4

Matthew Parker

Debbie Liske

Teena Thankachan

Harvard University

Spring 2022

Assignment Details 1

Initial Setup 2

Running the Simulation 6

Install OpenCV Applications 8

QUADROTOR AUTONOMOUS FLIGHT WITH PX4 AND MAVSDK 9

Testing and Conclusion 14

Appendix 1: Source Code 15

2. Initial Setup

Following provided instructions to complete the initial setup:
INSTALL THE PX4 FLIGHT STACK AND JMAVSIM & GAZEBO SIMULATORS

Source ubuntu sim files:

```
matt@matt-VirtualBox:~$ sudo wget https://raw.githubusercontent.com/PX4/Devguide
/v1.9.0/build_scripts/ubuntu_sim.sh
--2022-04-18 20:10:28--  https://raw.githubusercontent.com/PX4/Devguide/v1.9.0/b
uild_scripts/ubuntu_sim.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.1
33, 185.199.111.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1576 (1.5K) [text/plain]
Saving to: ‘ubuntu_sim.sh’

ubuntu_sim.sh       100%[===================>]   1.54K  --.-KB/s    in 0s

2022-04-18 20:10:28 (25.6 MB/s) - ‘ubuntu_sim.sh’ saved [1576/1576]
```

And then install:

```
matt@matt-VirtualBox:~$ source ubuntu_sim.sh
Downloading dependent script 'ubuntu_sim_common_deps.sh'
--2022-04-18 20:11:51--  https://raw.githubusercontent.com/PX4/Devguide/v1.9.0/b
uild_scripts/ubuntu_sim_common_deps.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.1
33, 185.199.109.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2839 (2.8K) [text/plain]
Saving to: ‘STDOUT’

-                   100%[===================>]   2.77K  --.-KB/s    in 0.001s

2022-04-18 20:11:52 (3.87 MB/s) - written to stdout [2839/2839]
```

COMPILE AND RUN THE GAZEBO SIMULATOR

Install some missing packages:

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install kconfiglib
Defaulting to user installation because normal site-packages is not writeable
Collecting kconfiglib
  Downloading kconfiglib-14.1.0-py2.py3-none-any.whl (145 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 145.9/145.9 KB 1.7 MB/s eta 0:00:00
Installing collected packages: kconfiglib
```

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install --user packaging
Collecting packaging
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.8/40.8 KB 1.1 MB/s eta 0:00:00
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/lib/python3/dist
-packages (from packaging) (2.4.6)
Installing collected packages: packaging
Successfully installed packaging-21.3
```

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install --user toml
Collecting toml
  Downloading toml-0.10.2-py2.py3-none-any.whl (16 kB)
Installing collected packages: toml
Successfully installed toml-0.10.2
```

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install --user jsonschema
Collecting jsonschema
  WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, stat
us=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectionPool(host
='files.pythonhosted.org', port=443): Read timed out. (read timeout=15)")': /pac
kages/55/b2/2c4af6a97c3f12c6d5a72b41d328c3996e14e1e46701df3fac1ed65119c9/jsonsch
ema-4.4.0-py3-none-any.whl
  Downloading jsonschema-4.4.0-py3-none-any.whl (72 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.7/72.7 KB 518.9 kB/s eta 0:00:00
Collecting pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0
  Downloading pyrsistent-0.18.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (119 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.8/119.8 KB 2.5 MB/s eta 0:00:00
Collecting importlib-resources>=1.4.0
  Downloading importlib_resources-5.7.1-py3-none-any.whl (28 kB)
```

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install future
Defaulting to user installation because normal site-packages is not writeable
Collecting future
  Downloading future-0.18.2.tar.gz (829 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 829.2/829.2 KB 3.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: future
```

```
matt@matt-VirtualBox:~/src/Firmware$ pip3 install gstreamer-player
Defaulting to user installation because normal site-packages is not writeable
Collecting gstreamer-player
  Downloading gstreamer-player-1.1.2.tar.gz (3.7 kB)
  Preparing metadata (setup.py) ... done
Collecting mutagen>=1.36.2
  Downloading mutagen-1.45.1-py3-none-any.whl (218 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 218.7/218.7 KB 2.0 MB/s eta 0:00:00
```

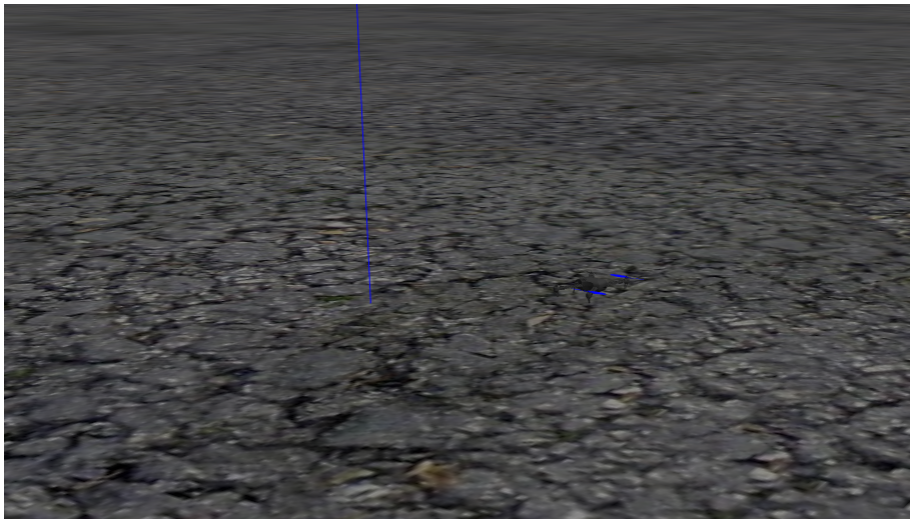
```
matt@matt-VirtualBox:~/src/Firmware$ sudo apt-get install libgstreamer-plugins-b
ad1.0-dev gstreamer1.0-plugins-good
Reading package lists... Done
Building dependency tree
Reading state information... Done
gstreamer1.0-plugins-good is already the newest version (1.16.2-1ubuntu2.1).
gstreamer1.0-plugins-good set to manually installed.
```

COMPILE AND RUN THE JMAVSIM SIMULATOR

Run the gazebo simulator:

```
matt@matt-VirtualBox:~/src/Firmware$ make px4_sitl gazebo
-- PX4 version: v1.13.0-alpha1-4618-g2e290345d3
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimu
m required is "3")
-- PX4 config file: /home/matt/src/Firmware/boards/px4/sitl/default.px4board
-- PLATFORM posix
-- ROMFSROOT px4fmu_common
-- TESTING y
-- ETHERNET y
-- PX4 config: px4_sitl_default
-- PX4 platform: posix
-- PX4 lockstep: enabled
-- cmake build type: RelWithDebInfo
-- The CXX compiler identification is GNU 9.4.0
-- The C compiler identification is GNU 9.4.0
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
```

All good after going back to install some of the missing dependencies:
The drone is successfully loaded into Gazebo:



INSTALL MAVSDK-Python LIBRARY

```
matt@matt-VirtualBox:~$ pip3 install mavsdk
Defaulting to user installation because normal site-packages is not writeable
Collecting mavsdk
  Downloading mavsdk-1.2.0-py3-none-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_17_x86_64.manylinux2014_x86_64.whl (11.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 2.4 MB/s eta 0:00:00
Collecting protobuf>=3.13.0
  Downloading protobuf-3.20.0-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.w
hl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 2.9 MB/s eta 0:00:00
```

UBUNTU QGROUNDCONTROL INSTALLATION

Initial configuration:

```
matt@matt-VirtualBox:~$ sudo usermod -a -G dialout $USER
[sudo] password for matt:
```

```
matt@matt-VirtualBox:~$ sudo apt-get remove modemmanager -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

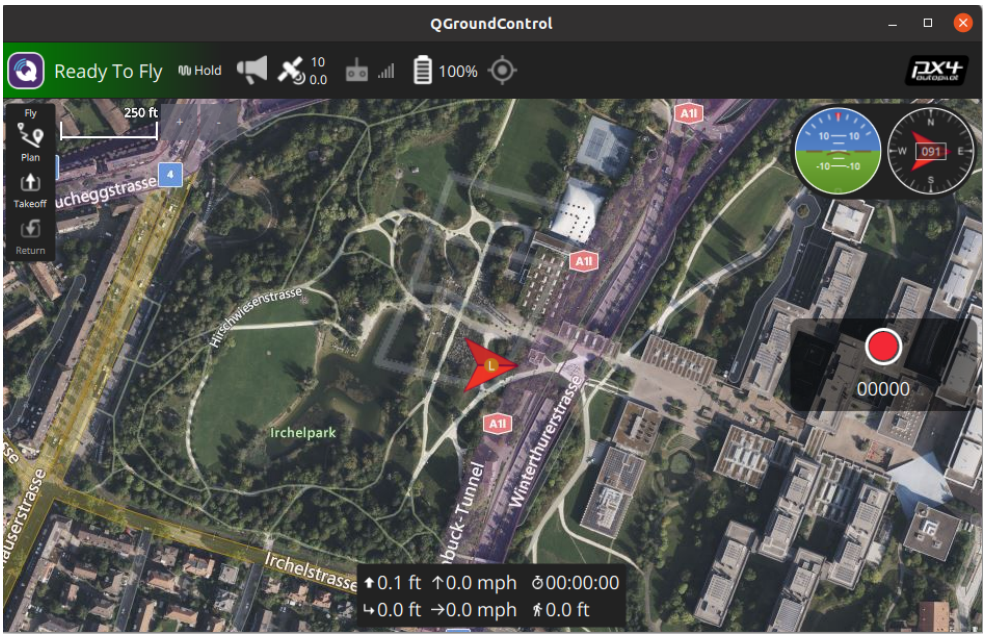
```
matt@matt-VirtualBox:~$ sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-l
ibav -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
```

Download from the website.
https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html

Go to downloads and make file executable:

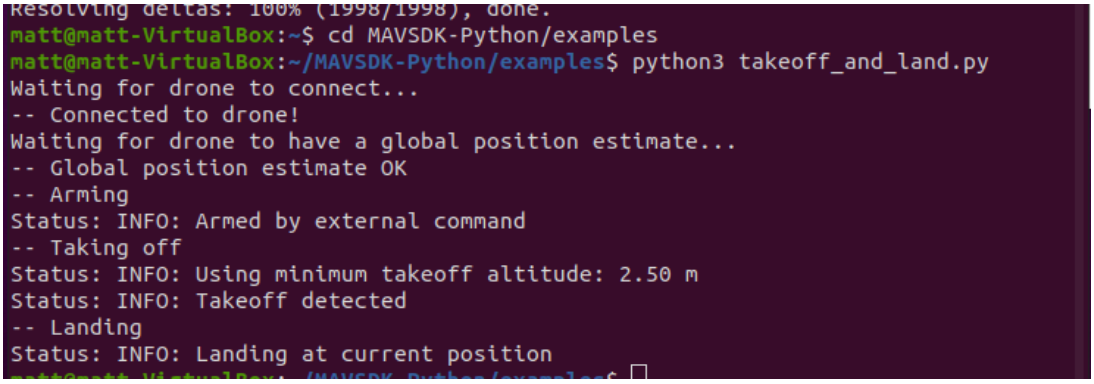
```
matt@matt-VirtualBox:~$ cd Downloads
matt@matt-VirtualBox:~/Downloads$ chmod +x ./QGroundControl.AppImage
```

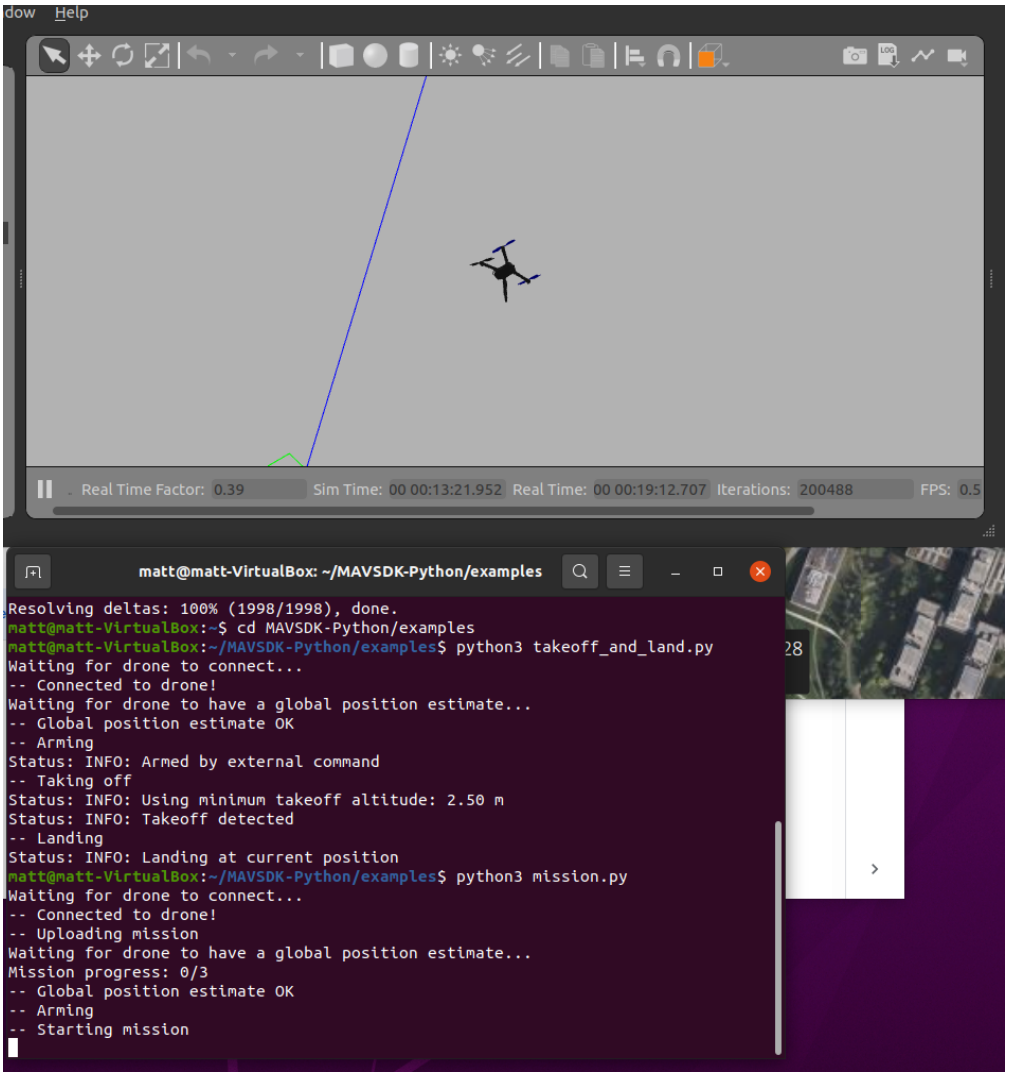
And run:



3. Running the Simulation

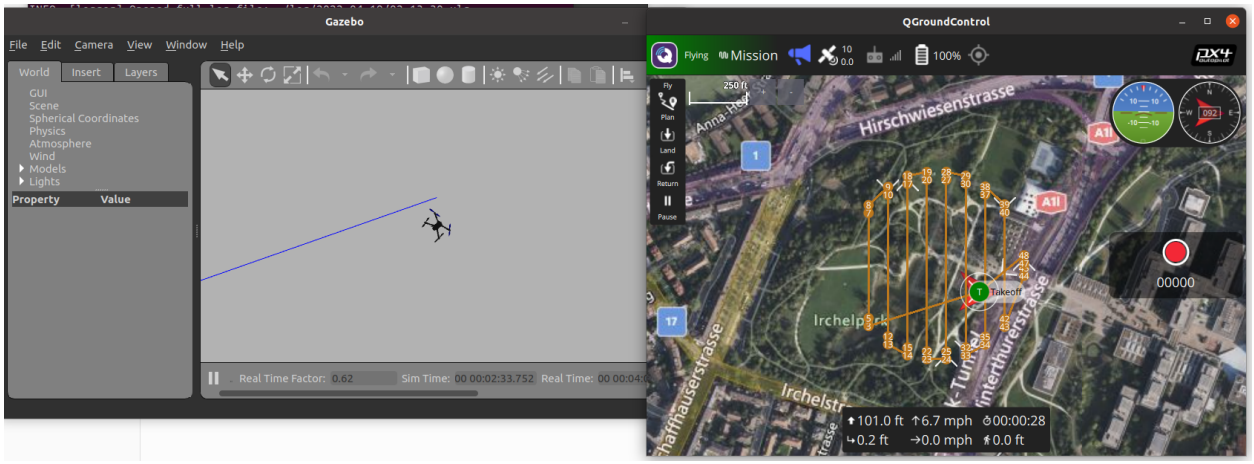
Verify some examples:

- Takeoff and Land:
  - 

- Running the mission.py example:
  - 

```
matt@matt-VirtualBox:~/MAVSDK-Python/examples$ python3 mission.py
Waiting for drone to connect...
-- Connected to drone!
-- Uploading mission
Waiting for drone to have a global position estimate...
Mission progress: 0/3
-- Global position estimate OK
-- Arming
-- Starting mission
Mission progress: 0/3
Mission progress: 0/3
Mission progress: 1/3
Mission progress: 1/3
Mission progress: 2/3
Mission progress: 3/3
matt@matt-VirtualBox:~/MAVSDK-Python/examples$
```

Attempting a basic trajectory from QGroundControl to ensure the drone is moving (Success!)



4. Install OpenCV Applications

INSTALL OPENCV
Update packages:

```
matt@matt-VirtualBox:~$ sudo apt update
[sudo] password for matt:
Hit:1 http://packages.ros.org/ros/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
```

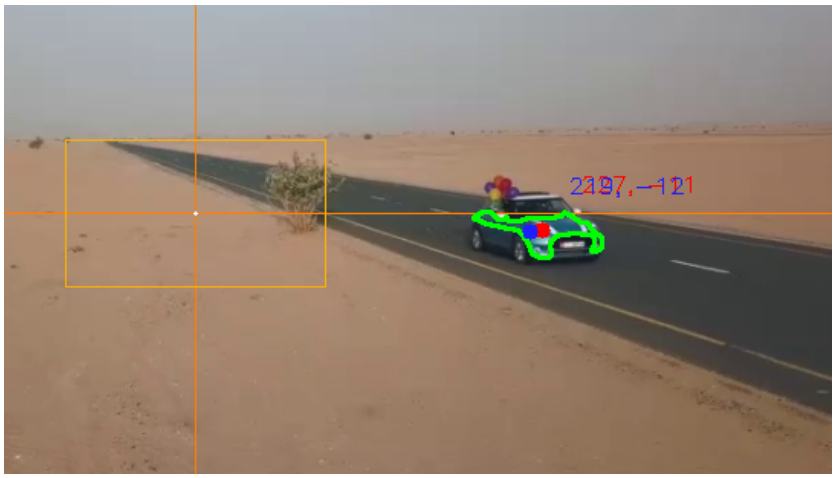
Install OpenCV app

```
matt@matt-VirtualBox:~$ sudo apt install python3-opencv
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-opencv is already the newest version (4.2.0+dfsg-5).
python3-opencv set to manually installed.
The following packages were automatically installed and are no longer required:
  libmbim-glib4 libmbim-proxy libqmi-glib5 libqmi-proxy usb-modeswitch
  usb-modeswitch-data
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 123 not upgraded.
```

TEST YOUR OPENCV INSTALLATION

Run the 'detect_test' setup first to confirm the camera and object tracking is working correctly.

```
matt@matt-VirtualBox:~/Downloads$ python3 detect_test.py
-- Camera opened successfully
params['scaling_factor']:  0.89
```



5. QUADROTOR AUTONOMOUS FLIGHT WITH PX4 AND MAVSDK

Run the following commands:

- cd ~/ # To change directory to your home directory
- Wget https://raw.githubusercontent.com/PX4/Devguide/v1.9.0/build_scripts/ubuntu_sim.sh
- source ubuntu_sim.sh

After the installation finishes, the directory will be automatically changed to: ~/src/Firmware

To compile the Gazebo simulator, run:

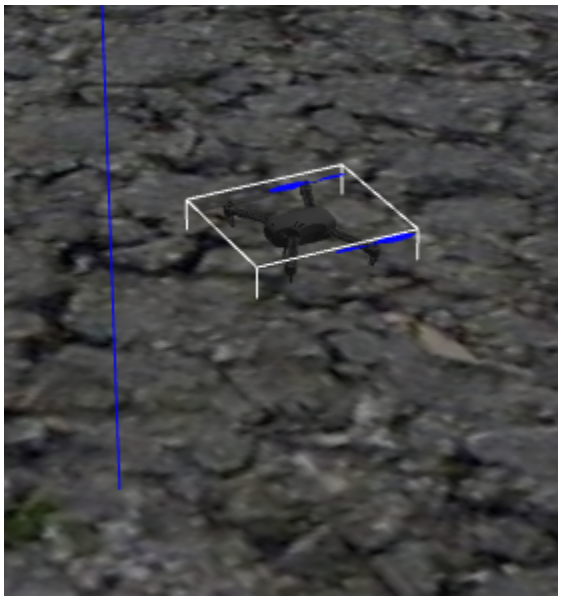
make px4_sitl gazebo

```
Creating symlink /home/teena/src/Firmware/build/px4_sitl_default/etc -> /home/teena/src/Firmware/build/px4_sitl_default


 _____  __   __    ___
| ___ \ \ \ / /   /   |
| |_/ /  \ V /   / /| |
|  __/   /   \  / /_| |
| |     / /^\ \ \___  |
\_|     \/   \/     |_/

px4 starting.

INFO  [px4] Calling startup script: /bin/sh etc/init.d-posix/rcS 0
INFO  [init] found model autostart file as SYS_AUTOSTART=10016
INFO  [param] selected parameter default file eeprom/parameters_10016
INFO  [parameters] BSON document size 597 bytes, decoded 597 bytes (INT32:17, FLOAT:12)
[param] Loaded: eeprom/parameters_10016
INFO  [dataman] data manager file './dataman' size is 7866640 bytes
PX4 SIM HOST: localhost
INFO  [simulator] Waiting for simulator to accept connection on TCP port 4560
INFO  [simulator] Simulator connected on TCP port 4560.
INFO  [commander] LED: open /dev/led0 failed (22)
INFO  [init] Mixer: etc/mixers/quad_w.main.mix on /dev/pwm_output0
INFO  [init] setting PWM_AUX_OUT none
```



7. INSTALL MAVSDK-Python LIBRARY

Check the official installation guide at: https://github.com/mavlink/MAVSDK-Python

To install MAVSDK-Python, simply run:

```
pip3 install mavsdk
```

To clone the MAVSDK-Python repository with the basic examples, simply run:

```
cd ~/  # To download the repository in your home directory
git clone https://github.com/mavlink/MAVSDK-Python.git
```

```
eena@teena-VirtualBox:~/MAVSDK-Python$ ls -ltr
otal 72
rw-rw-r-- 1 teena teena 5748 Apr 24 23:34 README.md
rw-rw-r-- 1 teena teena    7 Apr 24 23:34 MAVSDK_SERVER_VERSION
rw-rw-r-- 1 teena teena   52 Apr 24 23:34 MANIFEST.in
rw-rw-r-- 1 teena teena 1494 Apr 24 23:34 LICENSE.txt
rwxrwxr-x 2 teena teena 4096 Apr 24 23:34 examples
rw-rw-r-- 1 teena teena  114 Apr 24 23:34 tox.ini
rwxrwxr-x 2 teena teena 4096 Apr 24 23:34 tests
rw-rw-r-- 1 teena teena 5170 Apr 24 23:34 setup.py
rw-rw-r-- 1 teena teena   65 Apr 24 23:34 setup.cfg
rw-rw-r-- 1 teena teena   93 Apr 24 23:34 requirements.txt
rw-rw-r-- 1 teena teena   14 Apr 24 23:34 requirements-test.txt
rw-rw-r-- 1 teena teena   43 Apr 24 23:34 requirements-docs.txt
rw-rw-r-- 1 teena teena   46 Apr 24 23:34 requirements-dev.txt
rwxrwxr-x 2 teena teena 4096 Apr 24 23:34 proto
rwxrwxr-x 5 teena teena 4096 Apr 24 23:34 other
rwxrwxr-x 4 teena teena 4096 Apr 24 23:34 mavsdk
```

8. UBUNTU QGROUNDCONTROL INSTALLATION

Check the official installation guide at:
https://docs.qgroundcontrol.com/en/getting_started/download_and_install.html

Before installing QGroundControl for the first time:

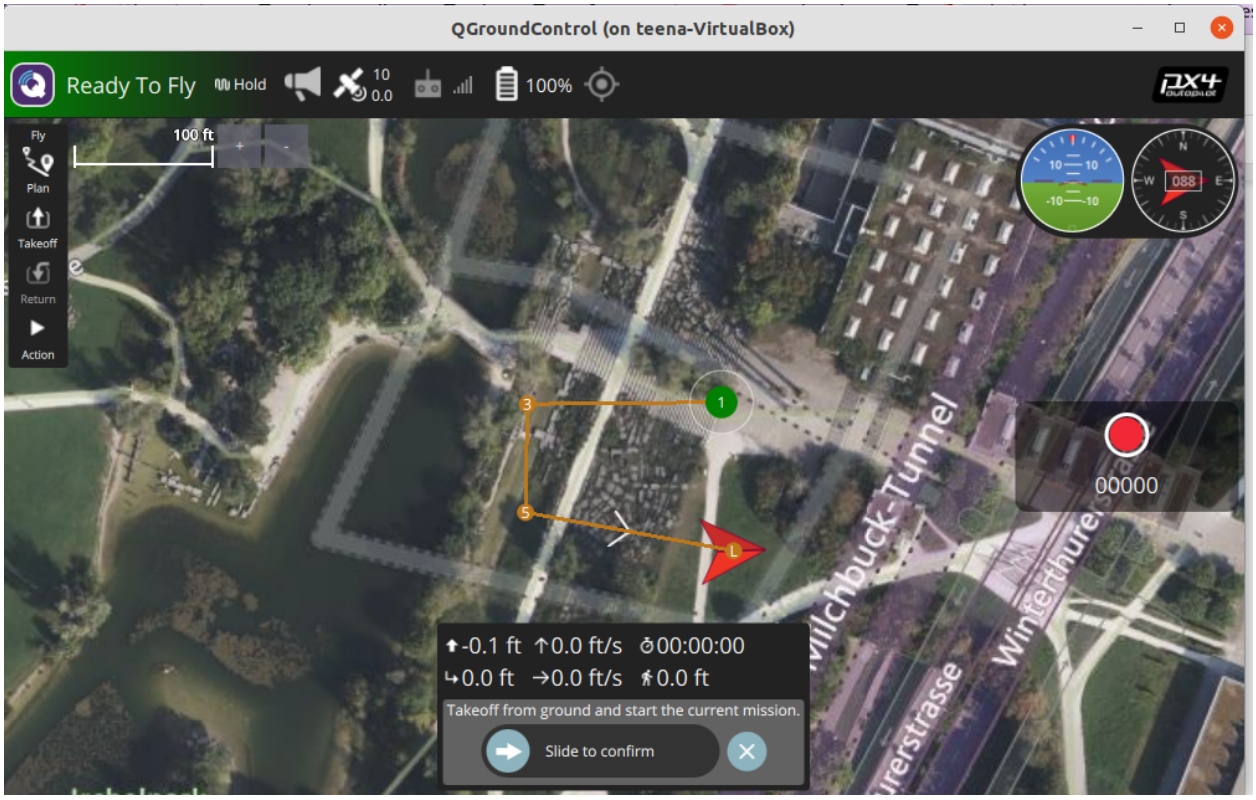
1. On the command prompt enter:
    sudo usermod -a -G dialout $USER
    sudo apt-get remove modemmanager -y
    sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav -y

Logout and login again to enable the change to user permissions.

2. To install QGroundControl for Ubuntu Linux 16.04 LTS or later:
    - Download QGroundControl.AppImage.
    - Install (and run) using the terminal commands:
      chmod +x ./QGroundControl.AppImage
      ./QGroundControl.AppImage (or double click)

```
teena@teena-VirtualBox:~/Downloads$ ls -ltr
total 165440
drwxrwxr-x 5 teena teena     4096 Mar  7 17:01  ros_gazebo_rviz
-rw-rw-r-- 1 teena teena     8529 Mar 19 07:43  ros_gazebo_rviz.zip
-rw-rw-r-- 1 teena teena 84697064 Apr 24 23:25  QGroundControl.AppImage
```

```
teena@teena-VirtualBox:~/Downloads$ chmod +x ./QGroundControl.AppImage
teena@teena-VirtualBox:~/Downloads$ ./QGroundControl.AppImage
Settings location "/home/teena/.config/QGroundControl.org/QGroundControl.ini" Is writable?: true
Filter rules "*Log.debug=false\nGStreamerAPILog.debug=true\nqt.qml.connections=false"
System reported locale: QLocale(English, Latin, United States) ; Name "en_US" ; Preffered (used in ma
ps):  "en-US"
VideoReceiverLog: Stop called on empty URI
VideoReceiverLog: Stop called on empty URI
MAVLinkLogManagerLog: MAVLink logs directory: "/home/teena/Documents/QGroundControl/Logs"
Map Cache in: "/home/teena/.cache/QGCMapCache300" / "qgcMapCache.db"
qml: QGCCorePlugin(0x55c75767f600) []
setCurrentPlanViewSeqNum
setCurrentPlanViewSeqNum
```



11. RUN IN SIMULATION THE EXAMPLE PRESENTED IN THE ARTICLE

* On command terminal window 1 run:

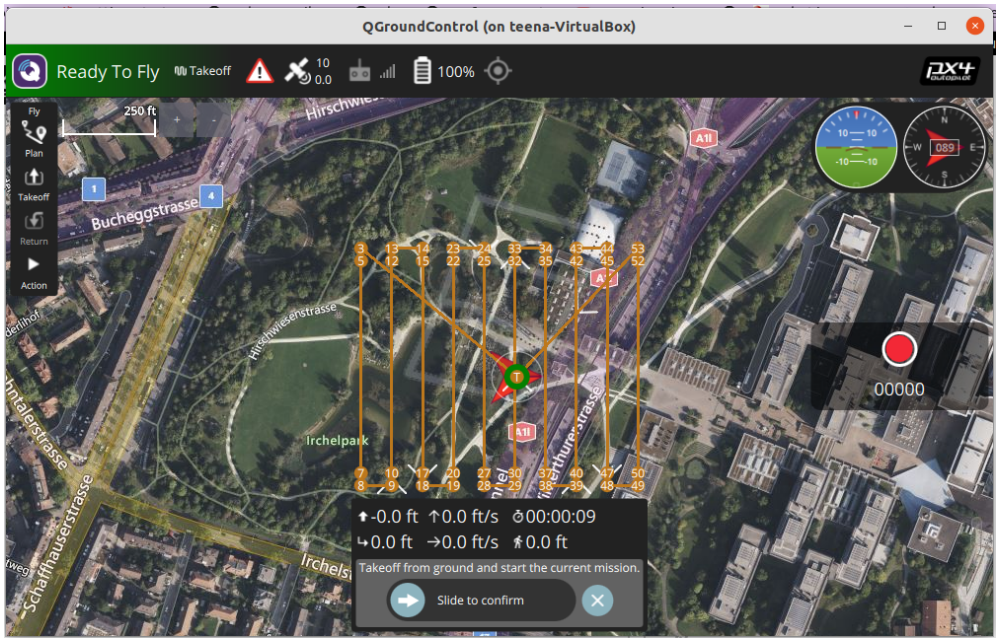
    cd ~/src/Firmware

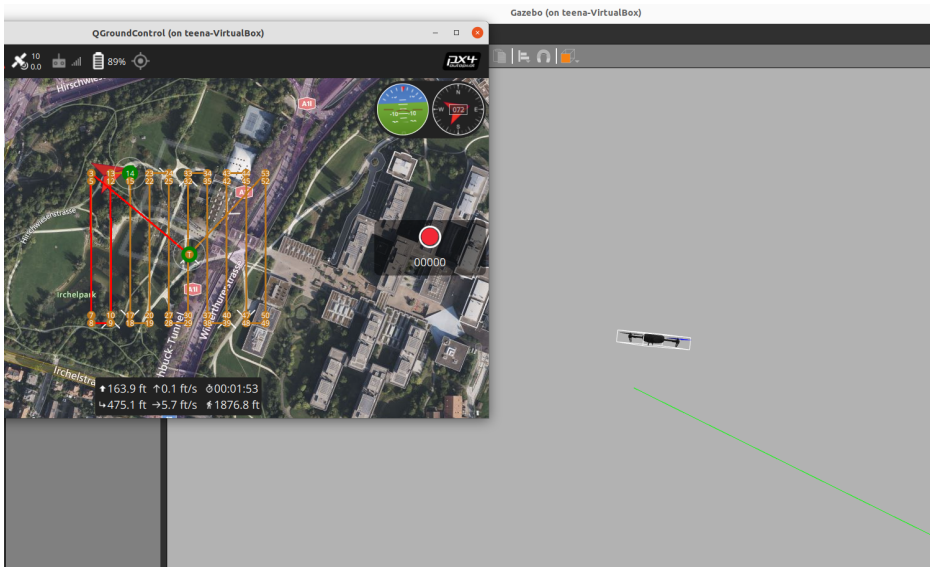
    make px4_sitl gazebo

    Open QGroundControl software.

Path Planning :

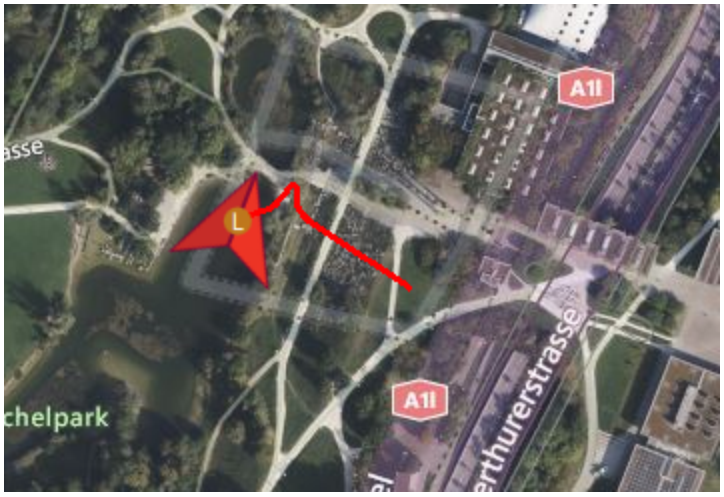
    Uploaded a basic survey plan on the drone:





Running drone from python commands :

```
teena@teena-VirtualBox:~/MAVSDK-Python/examples$ python3 follow_me_example.py
Waiting for drone to connect...
-- Connected to drone!
Waiting for drone to have a global position estimate...
-- Global position estimate OK
-- Arming
-- Taking Off
-- Starting Follow Me Mode
-- Following Target
-- Following Target
-- Following Target
-- Stopping Follow Me Mode
-- Landing
```



6. Testing Autonomous Control

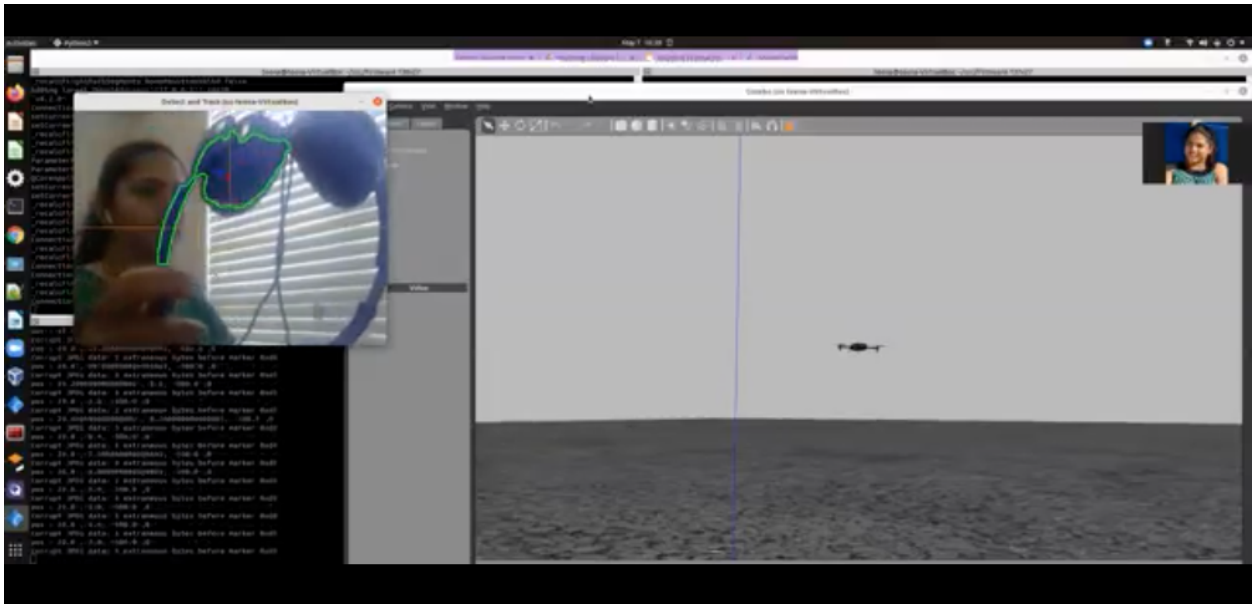
To test the code example in simulation, take the following steps:

1. First, run the PX4 & Gazebo, or PX4 & JMAVSim simulator.
2. Second, open QGroundControl.
3. Third, run the *track_and_follow.py* script.

If the camera detects a blue object within the HSV color range defined in the code, you will see the drone moving over QGroundControl's map in the direction of the detected target. Furthermore, if you move the blue target in front of the camera, the drone will move in the same direction, tracking the target only when it is outside the center rectangle. The *run_track_and_follow.md* text file summarizes all detailed steps to run the code in simulation. The figure below shows a screen capture of the *track_and_follow.py* script running.

Commands used to run

- make px4_sitl gazebo
- ./QGroundControl.AppImage
- python3 *track_and_follow.py*



7. Appendix 1: Source Code

Git Link:
https://github.com/debbieliske/DGMD_Robotics.git

A description of each file is provided below, with the full source code copied to the appendix:

1. Detect and Test
    a. detect_test.py file
    b. This code uses a previously recorded video in lieu of the camera input to verify the object detection and positioning software is functioning. The video is a blue car driving down the road, and the code will detect and follow the car as it moves through the frame.
2. Track and Follow
    a. track_and_follow_edit.py file
    b. This is the main function for the autonomous drone operations. Once the drone is flying and is put into autonomous mode, the program will import images from the camera, detect objects, and send drone steering commands to follow the position of the object.
3.

1. **Track and Follow Python Code**

```
#!/usr/bin/env python3
'''
'track_and_follow.py'

The drone follows an object detected by using color range segmentation with OpenCV.

Author: Raul Alvarez-Torrico (raul@tecbolivia.com)

This code example is based in part on the 'offboard_position_ned.py' and the
'telemetry_takeoff_and_land.py'
examples included in the MAVSDK-Python repository (see the 'examples' folder).

For an introduction to MAVSDK development with SITL simulation and all things related to
MAVSDK,
PX4 SITL simulation and Python asynchronous programming with the ‘asyncio’ library, please
refer
to my previous "Circuit Cellar" article "Quadrotor Autonomous Flight with PX4 and MAVSDK"
(Circuit Cellar [number], [month] 2019)
'''

import asyncio
import cv2
from mavsdk import System
#from mavsdk import (OffboardError, PositionNedYaw)

from mavsdk.offboard import PositionNedYaw as PositionNedYaw
from mavsdk.offboard import OffboardError as OffboardError
# Constants
PERCENT_CENTER_RECT  = 0.20 # For calculating the center rectangle's size
PERCENT_TARGET_RADIUS = 0.25 * PERCENT_CENTER_RECT # Minimum target radius to
follow
HOVERING_ALTITUDE    = 500.0 # Altitude in meters to which the drone will perform its tasks
NUM_FILT_POINTS      = 20 # Number of filtering points for the Moving Average Filter
DESIRED_IMAGE_HEIGHT = 480 # A smaller image makes the detection less CPU intensive

# A dictionary of two empty buffers (arrays) for the Moving Average Filter
filt_buffer = {'width':[], 'height':[]}

# A dictionary of general parameters derived from the camera image size,
# which will be populated later with the 'get_image_params' function
params = {'image_height':None, 'image_width': None,'resized_height':None,'resized_width':
None,
```

```
    'x_ax_pos':None, 'y_ax_pos':None, 'cent_rect_half_width':None, 'cent_rect_half_height':
None,
    'cent_rect_p1': None, 'cent_rect_p2': None, 'scaling_factor':None, 'min_tgt_radius':None}


### ---------- This is the application's 'main' asynchronous function ----------
async def run():
    """ Detects a target by using color range segmentation and follows it
    by using Offboard control and position NED coordinates. """

    # Open the video camera
    vid_cam = cv2.VideoCapture(0)

    # Let the camera warm up
    # await asyncio.sleep(2)

    # Check if the camera opened correctly
    if vid_cam.isOpened() is False:
        print('[ERROR] Couldnt open the camera.')
        return

    print('-- Camera opened successfully')

    # Compute general parameters
    await get_image_params(vid_cam)
    print(f"-- Original image width, height: {params['image_width']}, {params['image_height']}")

    # Get a reference to a 'System' object, which represents the drone,
    # and open a connection to it. The system address used here is the default
    # address for a simulated drone running in the same machine machine where
    # this code will run (localhost)
    drone = System()
    await drone.connect(system_address="udp://:14540") # To run with SITL simulation

    # ------ To run the code with a real drone connected to the PC via telemetry modules ------ #
    # await drone.connect(system_address="serial:///dev/ttyUSB0:57600")
    # CAUTION: Color range segmentation is not the best approach to detect and track objects
    #          This example is just a didactic proof of concept. Don't run it with a real drone
    #          unless you have good experience flying real drones and know what you are doing.
    # ------------------------------------------------------------------------------------------ #

    # Asynchronously poll the connection state until receiving an 'is_connected' confirmation
    print("Waiting for drone to connect...")
    async for state in drone.core.connection_state():
```

```
        print(f"Drone discovered with is_connected: {state.is_connected}")
        if state.is_connected:
            #print(f"Drone discovered with UUID: {state.uuid}")
            break

    # Activate the drone motors
    print("-- Arming")
    await drone.action.arm()

    # Send an initial position to the drone before changing to "offboard" flight mode
    print("-- Setting initial setpoint")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, 0.0, 0.0))

    # Change flight mode to "offboard", if it fails, disarm the motors and abort the script
    print("-- Starting offboard")
    try:
        await drone.offboard.start()
    except OffboardError as error:
        print(f"Starting offboard mode failed with error code: {error._result.result}")
        print("-- Disarming")
        await drone.action.disarm()
        return

    # Variables to store the NED coordinates, plus Yaw angle. Default pose:
    N_coord = 0
    E_coord = 0
    D_coord = -HOVERING_ALTITUDE # The drone will always detect and track at
HOVERING_ALTITUDE
    yaw_angle = 0 # Drone always points to North

    print("-- Taking off")
    await drone.action.takeoff()


    print("-- Go 0m North, 0m East, -5m Down within local coordinate system")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, -5.0, 0.0))
    await asyncio.sleep(10)

    # Make the drone go (take off) to the default pose
    print("-- Drone set pos")
    await drone.offboard.set_position_ned(
        PositionNedYaw(N_coord, E_coord, D_coord, yaw_angle))
    await asyncio.sleep(4) # Give the drone time to gain altitude
```

```
    # Infinite detect-follow loop
    while True:

        # Get the target coordinates (if any target was detected)
        tgt_cam_coord, frame, contour = await get_target_coordinates(vid_cam)

        # If a target was found, filter their coordinates
        if tgt_cam_coord['width'] is not None and tgt_cam_coord['height'] is not None:
            # Apply Moving Average filter to target camera coordinates
            tgt_filt_cam_coord = await moving_average_filter(tgt_cam_coord)

        # No target was found, set target camera coordinates to the Cartesian origin,
        # so the drone doesn't move
        else:
            # The Cartesian origin is where the x and y Cartesian axes are located
            # in the image, in pixel units
            tgt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']} # Needed just
for drawing objects
            tgt_filt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']}

        # Convert from camera coordinates to Cartesian coordinates (in pixel units)
        tgt_cart_coord = {'x':(tgt_filt_cam_coord['width'] - params['y_ax_pos']),
                          'y':(params['x_ax_pos'] - tgt_filt_cam_coord['height'])}

        # Compute scaling conversion factor from camera coordinates in pixel units
        # to Cartesian coordinates in meters
        COORD_SYS_CONV_FACTOR = 0.1

        # If the target is outside the center rectangle, compute North and East coordinates
        if abs(tgt_cart_coord['x']) > params['cent_rect_half_width'] or \
        abs(tgt_cart_coord['y']) > params['cent_rect_half_height']:
            # Compute North, East coordinates applying "camera pixel" to Cartesian conversion
factor
            E_coord = tgt_cart_coord['x'] * COORD_SYS_CONV_FACTOR
            N_coord = tgt_cart_coord['y'] * COORD_SYS_CONV_FACTOR
            # D_coord, yaw_angle don't change

        # Command the drone to the current NED + Yaw pose
        await drone.offboard.set_position_ned(
            PositionNedYaw(N_coord, E_coord, D_coord, yaw_angle))
        print(f"pos : {N_coord} ,{E_coord}, {D_coord} ,{yaw_angle}")
        # Draw objects over the detection image frame just for visualization
        frame = await draw_objects(tgt_cam_coord, tgt_filt_cam_coord, frame, contour)
```

```
        # Show the detection image frame on screen
        cv2.imshow("Detect and Track", frame)

        # Catch aborting key from computer keyboard
        key = cv2.waitKey(1) & 0xFF
        # If the 'q' key is pressed, break the 'while' infinite loop
        if key == ord("q"):
            break

    # After leaving the infinite loop, return the drone to home before ending the script
    print("-- Return to launch...")
    # await drone.action.return_to_launch()
    await drone.action.land()
    print("NOTE: check the drone has landed already before running again this script.")
    await asyncio.sleep(5) # Wait some time while the drone executes last command


async def get_image_params(vid_cam):
    """ Computes useful general parameters derived from the camera image size."""

    # Grab a frame and get its size
    is_grabbed, frame = vid_cam.read()
    params['image_height'], params['image_width'], _ = frame.shape

    # Compute the scaling factor to scale the image to a desired size
    if params['image_height'] != DESIRED_IMAGE_HEIGHT:
        params['scaling_factor'] = round((DESIRED_IMAGE_HEIGHT / params['image_height']), 2)
# Rounded scaling factor

    else:
        params['scaling_factor'] = 1

    print("params['scaling_factor']: ", params['scaling_factor'])

    # Compute resized width and height and resize the image
    params['resized_width'] = int(params['image_width'] * params['scaling_factor'])
    params['resized_height'] = int(params['image_height'] * params['scaling_factor'])
    dimension = (params['resized_width'], params['resized_height'])
    frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

    # Compute the center rectangle's half width and half height
    params['cent_rect_half_width'] = round(params['resized_width'] * (0.5 *
PERCENT_CENTER_RECT)) # Use half percent (0.5)
```

```
    params['cent_rect_half_height'] = round(params['resized_height'] * (0.5 *
PERCENT_CENTER_RECT)) # Use half percent (0.5)

    # Compute the minimum target radius to follow. Smaller detected targets will be ignored
    params['min_tgt_radius'] = round(params['resized_width'] * PERCENT_TARGET_RADIUS)

    # Compute the position for the X and Y Cartesian coordinates in camera pixel units
    params['x_ax_pos'] = int(params['resized_height']/2 - 1)
    params['y_ax_pos'] = int(params['resized_width']/2 - 1)

    # Compute two points: p1 in the upper left and p2 in the lower right that will be used to
    # draw the center rectangle iin the image frame
    params['cent_rect_p1'] = (params['y_ax_pos'] - params['cent_rect_half_width'],
                              params['x_ax_pos'] - params['cent_rect_half_height'])
    params['cent_rect_p2'] = (params['y_ax_pos'] + params['cent_rect_half_width'],
                              params['x_ax_pos'] + params['cent_rect_half_height'])

    return


async def get_target_coordinates(vid_cam):
    """ Detects a target by using color range segmentation and returns its 'camera pixel'
coordinates."""

    # Use the 'threshold_inRange.py' script included with the code to get
    # your own bounds with any color
    # To detect a blue target:
    HSV_LOWER_BOUND = (107, 119, 41)
    HSV_UPPER_BOUND = (124, 255, 255)

    # Grab a frame in BGR (Blue, Green, Red) space color
    is_grabbed, frame = vid_cam.read()

    # Resize the image frame for the detection process, if needed
    if params['scaling_factor'] != 1:
        dimension = (params['resized_width'], params['resized_height'])
        frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

    # Blur the image to remove high frequency content
    blurred = cv2.GaussianBlur(frame, (11, 11), 0)

    # Change color space from BGR to HSV
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

```
    # Histogram equalisation to minimize the effect of variable lighting
    # hsv[:, :, 0] = cv2.equalizeHist(hsv[:, :, 0]) # on the H-channel
    # hsv[:, :, 1] = cv2.equalizeHist(hsv[:, :, 1]) # on the S-channel
    # hsv[:, :, 2] = cv2.equalizeHist(hsv[:, :, 2]) # on the V-channel

    # Get a mask with all the pixels inside our defined color boundaries
    mask = cv2.inRange(hsv, HSV_LOWER_BOUND, HSV_UPPER_BOUND)

    # Erode and dilate to remove small blobs
    mask = cv2.erode(mask, None, iterations=3)
    mask = cv2.dilate(mask, None, iterations=3)

    # Find all contours in the masked image
    #tt_, contours, _ = cv2.findContours(mask,  cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    #ttcontours, hierarchy= cv2.findContours(mask,  cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours , _ = cv2.findContours(mask,  cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Centroid coordinates to be returned:
    cX = None
    cY = None

    # To save the larges contour, presumably the detected object
    largest_contour = None

    # Check if at least one contour was found
    if len(contours) > 0:
        # Get the largest contour of all posibly detected
        largest_contour = max(contours, key=cv2.contourArea)

        # Compute the radius of an enclosing circle aorund the largest contour
        ((x, y), radius) = cv2.minEnclosingCircle(largest_contour)

        # Compute centroid only if contour radius is larger than 0.5 half the center rectangle
        if radius > params['min_tgt_radius']:
            # Compute contour raw moments
            M = cv2.moments(largest_contour)
            # Get the contour's centroid
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])

    # Return centroid coordinates (camera pixel units), the analized frame and the largest contour
```

```
    return {'width':cX, 'height':cY}, frame, largest_contour


async def moving_average_filter(coord):
    """ Applies Low-Pass Moving Average Filter to a pair of (x, y) coordinates."""

    # Append new coordinates to filter buffers
    filt_buffer['width'].append(coord['width'])
    filt_buffer['height'].append(coord['height'])

    # If the filters were full already with a number of NUM_FILT_POINTS values,
    # discard the oldest value (FIFO buffer)
    if len(filt_buffer['width']) > NUM_FILT_POINTS:
        filt_buffer['width'] = filt_buffer['width'][1:]
        filt_buffer['height'] = filt_buffer['height'][1:]

    # Compute filtered camera coordinates
    N = len(filt_buffer['width']) # Get the number of values in buffers (will be <
NUM_FILT_POINTS at the start)

    # Sum all values for each coordinate
    w_sum = sum( filt_buffer['width'] )
    h_sum = sum( filt_buffer['height'] )
    # Compute the average
    w_filt = int(round(w_sum / N))
    h_filt = int(round(h_sum / N))

    # Return filtered coordinates as a dictionary
    return {'width':w_filt, 'height':h_filt}


async def draw_objects(cam_coord, filt_cam_coord, frame, contour):
    """ Draws visualization objects from the detection process.
    Position coordinates of every object are always in 'camera pixel' units"""

    # Draw the Cartesian axes
    cv2.line(frame, (0, params['x_ax_pos']), (params['resized_width'], params['x_ax_pos']), (0,
128, 255), 1)
    cv2.line(frame, (params['y_ax_pos'], 0), (params['y_ax_pos'], params['resized_height']), (0,
128, 255), 1)
    cv2.circle(frame, (params['y_ax_pos'], params['x_ax_pos']), 1, (255, 255, 255), -1)

    # Draw the center (tolerance) rectangle
    cv2.rectangle(frame, params['cent_rect_p1'], params['cent_rect_p2'], (0, 178, 255), 1)
```

```
    # Draw the detected object's contour, if any
    cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)

    # Compute Cartesian coordinates of unfiltered detected object's centroid
    x_cart_coord = cam_coord['width'] - params['y_ax_pos']
    y_cart_coord = params['x_ax_pos'] - cam_coord['height']

    # Compute Cartesian coordinates of filtered detected object's centroid
    x_filt_cart_coord = filt_cam_coord['width'] - params['y_ax_pos']
    y_filt_cart_coord = params['x_ax_pos'] - filt_cam_coord['height']

    # Draw unfiltered centroid as a red dot, including coordinate values
    cv2.circle(frame, (cam_coord['width'], cam_coord['height']), 5, (0, 0, 255), -1)
    cv2.putText(frame, str(x_cart_coord) + ", " + str(y_cart_coord),
        (cam_coord['width'] + 25, cam_coord['height'] - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 255), 1)

    # Draw filtered centroid as a blue dot, including coordinate values
    cv2.circle(frame, (filt_cam_coord['width'], filt_cam_coord['height']), 5, (255, 30, 30), -1)
    cv2.putText(frame, str(x_filt_cart_coord) + ", " + str(y_filt_cart_coord),
        (filt_cam_coord['width'] + 25, filt_cam_coord['height'] - 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 30, 30), 1)

    return frame # Return the image frame with all drawn objects




if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(run())
```

2. OpenCV Python Code

```
### ---------- This is the application's 'main' asynchronous function ----------
async def run():
    """ Detects a target by using color range segmentation and follows it
    by using Offboard control and position NED coordinates. """

    vid_cam = cv2.VideoCapture(0)

    if vid_cam.isOpened() is False:
```

```
        print('[ERROR] Couldnt open the camera.')
        return
    print('-- Camera opened successfully')

    await get_image_params(vid_cam)
    print(f"-- Original image width, height: {params['image_width']}, {params['image_height']}")

    drone = System()
    await drone.connect(system_address="udp://:14540")
    # ... Additional drone initialization code lines go here...

    N_coord = 0
    E_coord = 0
    D_coord = -HOVERING_ALTITUDE
    yaw_angle = 0

    await drone.offboard.set_position_ned(PositionNedYaw(N_coord, E_coord, D_coord,
yaw_angle))
    await asyncio.sleep(4)

    while True:
        tgt_cam_coord, frame, contour = await get_target_coordinates(vid_cam)

        if tgt_cam_coord['width'] is not None and tgt_cam_coord['height'] is not None:
            tgt_filt_cam_coord = await moving_average_filter(tgt_cam_coord)
        else:
            tgt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']}
            tgt_filt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']}

        tgt_cart_coord = {'x':(tgt_filt_cam_coord['width'] - params['y_ax_pos']),
                          'y':(params['x_ax_pos'] - tgt_filt_cam_coord['height'])}

        COORD_SYS_CONV_FACTOR = 0.1

        if abs(tgt_cart_coord['x']) > params['cent_rect_half_width'] or \
        abs(tgt_cart_coord['y']) > params['cent_rect_half_height']:
            E_coord = tgt_cart_coord['x'] * COORD_SYS_CONV_FACTOR
            N_coord = tgt_cart_coord['y'] * COORD_SYS_CONV_FACTOR
            # D_coord, yaw_angle don't change

        await drone.offboard.set_position_ned(PositionNedYaw(N_coord, E_coord, D_coord,
yaw_angle))

        frame = await draw_objects(tgt_cam_coord, tgt_filt_cam_coord, frame, contour)
```

```
        cv2.imshow("Detect and Track", frame)

        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

    print("-- Return to launch...")
    await drone.action.return_to_launch()
    print("NOTE: check the drone has landed already before running again this script.")
    await asyncio.sleep(5)

async def get_image_params(vid_cam):
    """ Computes useful general parameters derived from the camera image size."""

    _, frame = vid_cam.read()
    params['image_height'], params['image_width'], _ = frame.shape

    if params['image_height'] != DESIRED_IMAGE_HEIGHT:
        params['scaling_factor'] = round((DESIRED_IMAGE_HEIGHT / params['image_height']), 2)
    else:
        params['scaling_factor'] = 1

    params['resized_width'] = int(params['image_width'] * params['scaling_factor'])
    params['resized_height'] = int(params['image_height'] * params['scaling_factor'])
    dimension = (params['resized_width'], params['resized_height'])
    frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

    params['cent_rect_half_width'] = round(params['resized_width'] * (0.5 *
PERCENT_CENTER_RECT))
    params['cent_rect_half_height'] = round(params['resized_height'] * (0.5 *
PERCENT_CENTER_RECT))

    params['min_tgt_radius'] = round(params['resized_width'] * PERCENT_TARGET_RADIUS)

    params['x_ax_pos'] = int(params['resized_height']/2 - 1)
    params['y_ax_pos'] = int(params['resized_width']/2 - 1)

    params['cent_rect_p1'] = (params['y_ax_pos']-params['cent_rect_half_width'],
                              params['x_ax_pos']-params['cent_rect_half_height'])
    params['cent_rect_p2'] = (params['y_ax_pos']+params['cent_rect_half_width'],
                              params['x_ax_pos']+params['cent_rect_half_height'])

    return
```

```
async def get_target_coordinates(vid_cam):
    """ Detects a target by using color range segmentation and returns its
        'camera pixel' coordinates."""

    HSV_LOWER_BOUND = (107, 119, 41)
    HSV_UPPER_BOUND = (124, 255, 255)

    _, frame = vid_cam.read()

    if params['scaling_factor'] != 1:
        dimension = (params['resized_width'], params['resized_height'])
        frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

    blurred = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, HSV_LOWER_BOUND, HSV_UPPER_BOUND)
    mask = cv2.erode(mask, None, iterations=3)
    mask = cv2.dilate(mask, None, iterations=3)

    _, contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    cX = None
    cY = None
    largest_contour = None

    if len(contours) > 0:
        largest_contour = max(contours, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(largest_contour)

        if radius > params['min_tgt_radius']:
            M = cv2.moments(largest_contour)
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])

    return {'width':cX, 'height':cY}, frame, largest_contour

async def moving_average_filter(coord):
    """ Applies Low-Pass Moving Average Filter to a pair of (x, y) coordinates."""

    filt_buffer['width'].append(coord['width'])
    filt_buffer['height'].append(coord['height'])
```

```
    if len(filt_buffer['width']) > NUM_FILT_POINTS:
        filt_buffer['width'] = filt_buffer['width'][1:]
        filt_buffer['height'] = filt_buffer['height'][1:]

    N = len(filt_buffer['width'])

    w_sum = sum( filt_buffer['width'] )
    h_sum = sum( filt_buffer['height'] )

    w_filt = int(round(w_sum / N))
    h_filt = int(round(h_sum / N))

    return {'width':w_filt, 'height':h_filt}


async def draw_objects(cam_coord, filt_cam_coord, frame, contour):
    """ Draws visualization objects from the detection process.
        Position coordinates of every object are always in 'camera pixel' units"""

    cv2.line(frame, (0, params['x_ax_pos']), (params['resized_width'], params['x_ax_pos']), (0,
128, 255), 1)
    cv2.line(frame, (params['y_ax_pos'], 0), (params['y_ax_pos'], params['resized_height']), (0,
128, 255), 1)
    cv2.circle(frame, (params['y_ax_pos'], params['x_ax_pos']), 1, (255, 255, 255), -1)

    cv2.rectangle(frame, params['cent_rect_p1'], params['cent_rect_p2'], (0, 178, 255), 1)

    cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)

    x_cart_coord = cam_coord['width'] - params['y_ax_pos']
    y_cart_coord = params['x_ax_pos'] - cam_coord['height']

    x_filt_cart_coord = filt_cam_coord['width'] - params['y_ax_pos']
    y_filt_cart_coord = params['x_ax_pos'] - filt_cam_coord['height']

    cv2.circle(frame, (cam_coord['width'], cam_coord['height']), 5, (0, 0, 255), -1)
    cv2.putText(frame, str(x_cart_coord) + ", " + str(y_cart_coord),
        (cam_coord['width'] + 25, cam_coord['height'] - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 255), 1)

    cv2.circle(frame, (filt_cam_coord['width'], filt_cam_coord['height']), 5, (255, 30, 30), -1)
    cv2.putText(frame, str(x_filt_cart_coord) + ", " + str(y_filt_cart_coord),
        (filt_cam_coord['width'] + 25, filt_cam_coord['height'] - 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 30, 30), 1)
```

return frame

####################################Teena################################